《程序设计与算法》——《计算导论与 C 语言基础》学习笔记

Du Ang
du2ang233@gmail.com

2017 年 6 月 12 日

1 课程介绍和二进制数

1.1 课程介绍

计算导论

- 计算机的基本原理
- 计算机的发展趋势
- 程序运行的基本原理

C 程序设计

- 感性认识 C 程序
- 理性认识 C 程序

1.2 计算机的基本原理

计算机的理论模型——图灵机 图灵机——判定一个问题是否可以计算。

对于一个问题的输入 A，A 是否能推证出 B？如果能找到一个图灵机，得出对应的符号序列 B，那么从 A 到 B 就是可计算的；否则，该问题不可计算。

图灵机的三大特点：简单、强大、可实现。

1.3 数的二进制表示

十进制整数转换为二进制整数：除以 2 取余数，除到商为 0 为止（“触底反弹”）。

二进制整数转换为十进制整数：要从右到左用二进制的每个数去乘以 2 的相应次方。

二进制转八进制：从右向左，每三位进行一次转换。

二进制转十六进制：从右向左，每四位进行一次转换。

1.4 二进制数的布尔运算

基本逻辑：与，或，非

复合逻辑：同或、异或，与非、或非，与或非

二进制数运算可以通过基本的布尔运算实现，而布尔运算都可以通过电路实现，所以电路能够计算。

2 计算机的发展和摩尔定律

2.1 计算机发展史

早期计算机

- 手工计算机
- 机械计算机
- 计算机原型

现代计算机

- 电子管计算机：ENIAC，EDVAC。EDVAC 是所有现代计算机的原型。
- 晶体管计算机
- 集成电路计算机
- 超大规模集成电路计算机
- 未来第五代计算机：量子计算机、生物计算机

2.2 摩尔定律

摩尔定律 芯片密度大约每 18 个月增加 1 倍；CPU 性能价格比大约每 18 个月翻一番。

摩尔定律下的挑战：

1. 散热
2. 晶体管大小限制
3. 电泄露

随着晶体管密度与速度的增加，芯片会消耗更多的电力，产生更多的热能。能不能把芯片做大以增加散热呢？答案是不能，更大的芯片意味着更高的电压，反而会带来更大的热量。

3 计算机结构

计算机结构 运算器、控制器、存储器、输入设备、输出设备，它们通过总线相连。

CPU 包含运算器、控制器和存储器的一部分（高速缓存 Cache）。

CPU 访问数据的局部性原理：

- 时间局部性：如果一个内存地址正在被使用，那么近期它很可能还会被再次访问。
- 空间局部性：在最近的将来可能用到的信息很可能与当前使用的信息是相邻的。

存储器包括 RAM（Ramdom Access Memory）和 ROM（Read Only Memory）两大类。ROM（掩膜 ROM）在生产时写入，用户不可更改。

RAM

- DRAM（Dynamic RAM）：必须周期性刷新以保持存储内容
  - EDO DRAM（Extended Data Out DRAM）扩展数据输出动态存储器
  - SDRAM（Synchronous DRAM）同步动态存储器
  - DDR（Double Data Rate SDRAM）双数据输出同步动态存储器，目前的主流内存
- SRAM（Static RAM）：不需要周期性刷新

ROM

- PROM（Programmable ROM）
- EPROM（Erasable PROM）
- EEPROM（Electrically EPROM）
- Flash EPROM

为什么 32 位计算机最大只能支持 4G 内存？因为 CPU 的寻址能力以字节（Byte）为单位，32 位计算机的寻址空间只有 $2^{32} = 4\text{GB}$。

4 编程语言和程序设计

4.1 关于编程语言的三个问题

问题 1 是不是无论我们在程序里写什么“单词”，计算机都能明白？

答 不是。编程语言定义了一些有特定含义的“关键字”，计算机“只能明白”这些“词”的含义。

问题 2 是不是无论我们在程序里写什么“数”和“计算符号”，计算机都能明白？

答 不是。计算机只能“看懂”某些类型的数据，这些“数据类型”和相应的“操作符号”是定义好的。

问题 3 世间用“程序来表达的逻辑”纷繁复杂，多少“句式”才能够用？

答 三种。顺序、分支、循环。

4.2 什么样的程序是“好程序”

在这门课中，重视：

- 正确性，结果对
- 可读性，能看懂
- 结构性，看着美

在这门课中，不重视：

- 少用了几个变量
- 程序行数少
- 程序运行快

5 结构化程序设计

在面临一个问题时，没有想出解决方案之前不要动手写程序。原因有三：

- 没有想出解决方案，不可能写出程序
- 急于写程序很可能会限制思维，导致想不出创新方案
- 急于写程序会导致程序漏洞百出，改来改去，结构混乱

结构化程序设计的基本思想：先粗后细，先抽象后具体。
写程序的过程：由大到小，由粗到精，由抽象到具体地方法分析。
程序的结构：若干个模块，高内聚、低耦合。

开始编写程序，我用 Ubuntu 上的文本编辑器来写，通过 g++ 编译。[1] 编译示例如下：

```
g++ helloworld.cpp -o helloworld
```

程序中 `int main(int argc, char* argv[])` 中的 `argc` 是参数的个数，`argv[]` 是参数值。[2]
`atoi` 函数可以将 `char` 型转换为 `int` 型，在 C++ 中要包含 `cstdlib`。

6 C/C++ 简介

Alan J. Perlis 在 1960 年写的 *Report on the Algorithmic Language ALGOL 60* 是程序设计语言领域一篇里程碑式的论文。

程序设计语言的构成：

1. 数据成分
2. 运算成分
3. 控制成分
4. 传输成分

C++ 语言支持了 C 语言所有的特性。

Assignment 2 C 语言中数组的初始长度能不能由用户输入来决定？在我学 C 语言时，老师通过 VC6.0 告诉我们不可以。但是现在我用 gcc（ver. 5.4）却可以编译成功，而且 gcc 没有对数组越界进行报错。具体来说，如果定义数组 `int a[n]`，初始长度由用户输入的 `n` 来决定，那么越界部分既可以读取，也可以写入；如果定义数组 `int b[2]`，即初始长度固定为某个整数，那么可以读取越界值 `b[2]`、`b[3]` 等，但是无法写入（运行程序时，向越界部分写入时会报 `stack smashing` 的错误）。读取到的越界部分的值均是没有初始化过的、内存中的随机值。gcc 支持的 C 标准和 VC6.0 支持的 C 标准有所不同：gcc 允许使用变量作为数组初始长度，不检查数组越界。[3]

示例代码如下：

[1] http://wiki.ubuntu.org.cn/Compiling_Cpp
[2] http://stackoverflow.com/questions/18649547/how-to-find-the-length-of-argv-in-c
[3] https://www.zhihu.com/question/28786159

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
    int n;
    scanf("%d", &n);
    int a[n];

    for (int i = 0; i < n*2; i++) {
        printf("a[%d] = %d\n", i, a[i]);
    }
    for (int i = 0; i < n*2; i++) {
        a[i] = -1;
        printf("a[%d] = %d\n", i, a[i]);
    }

    int b[2];
    for (int i = 0; i < 4; i++) {
        printf("b[%d] = %d\n", i, b[i]);
    }
    // b[2] = 3;    // stack smashing
    /*for (int i = 0; i < 4; i++) { // stack smashing
        b[i] = -2;
        printf("b[%d] = %d\n", i, b[i]);
    }*/

    return 0;
}
```

Assignment 3 `ceil(a)` 可以返回比 a 大的最小整数（`#include <math.h>`）。

7 C 语言的数据成分

1. 变量的定义。先定义，再使用。定义时最好赋初始值。
2. 整数型的分类。C 标准没有具体规定各种整型数据所占内存字节数，只要求 `long` 型不短于 `int` 型，`short` 型不长于 `int` 型。VC++ 中，`int == long int (4 Byte)`。`sizeof()` 可以打印指定类型占内存的字节数。
3. 正数的补码是它本身；负数的补码 = 反码 + 1。
4. 打印控制符 `hex`、`oct`、`dec`。十六进制数以 `0x` 开头，八进制数以 `0` 开头。可以通过控制打印输出来完成数制间的转换。注意：同一程序内，设定打印控制符后，之后所有的打印都会默认这个设定

好的打印控制符。

5. 计算机中二进制数的最高位是 1、其他位是 0 时，最高位既表示负号、也表示整数的最高位。因此 10…0 是能表示的最小的整数。

6. 浮点型 = 实型。float 型的有效精度为 7 位，double 型的有效精度为 15 位。setprecision(x) 设置 x 位打印精度（*#include <iomanip>*）。应避免将一个很大的数与一个很小的数相加或相减，否则就会“丢失”小的数。

7. 字符型占一个字节，存储和整型相同，可以与整型相互赋值、运算。char 型变量赋值时要使用单引号。

8. 布尔类型占一个字节，有“非 0 即 1”的特点。为什么不用一位？因为字节是计算机能控制的最小单元。

9. 常量：字面常量、符号常量。定义常量时，类型前加 const 限定符。字面常量（即赋值符号右边的具体值）也是有类型的，可以通过后缀来明确定义其类型，如 L、U、LU、F 等，均可以小写。浮点型常量默认为 double 类型。

10. 标识符：各种有效字符序列（名字），由字母、数字、下划线组成，第一个字符不能为下划线，不可与保留字冲突。变量的命令方法主要有两种：匈牙利命名法和驼峰命名法。合适的地方用合适的命名方法，不要死磕某一种。定义变量时最好加注释。

8 C 语言的运算成分

8.1 赋值运算符

`int a, b, c = 5;` 表示只给 c 赋初值

`int a = b = c = 5;` 错误，定义时不能连等

若“=”两边类型不一致，赋值时右边自动转换为左边的类型。

长数赋给短数，截取低位赋给短数。短数赋给长数，原来是什么数，现在还是什么数。计算机处理时会根据有无符号在高位补 0 或补 1，下面是 short 型赋给 long 型的例子。

- 若 short 型为无符号数：
  - short 型 16 位到 long 型低 16 位，long 型高 16 位补 0；
- 若 short 型为有符号数：
  - short 型 16 位到 long 型低 16 位；
  - 若 short 型最高位为 0，则 long 型高 16 位补 0；
  - 若 short 型最高位为 1，则 long 型高 16 位补 1；

signed 和 unsigned 之间原封不动地赋值，不管符号位还是数字位。

赋值表达式也是有返回值的，返回值即为 = 所传递的值。连续赋值运算，自右向左结合。

8.2 算术运算符和算术表达式

% 是模运算，操作数必须是整数。整数运算，结果仍然是整数，如 5 / 3 的结果为 1；实数运算，结果是 double 型，如 5.3 / 3 和 5 / 3.0 的结果都是 double 型。在算术运算中的优先级：double > long > unsigned > int > char, short；double > float。

8.3 自增自减运算符

(-i)++; 错误，++/-- 只能作用于变量，不能作用于表达式。

cout 语句中包含多个表达式时，从右向左计算表达式的值。

c = 2; (c++) + (++c) = ? 答案是 6，表达式中有 ++c，则先计算 ++c，此时 c 自增为 3，把 c = 3 代入 c++，得到最后的结果为 6。

8.4 关系运算符

混合运算的优先级：逻辑非（!）> 算术运算符 > 关系运算符 > 逻辑与（&&）> 逻辑或（||）> 赋值运算符。不确定优先级时，加括号。

逻辑表达式求解时，并不总是执行所有的运算。只有在必须执行下一个逻辑运算符才能求出表达式的解时，才执行该运算符。

示例程序：

```
int a = 0, b = 0;
a = 5 > 3 && 2 || 8 < 4 - (b = !0);
cout << a << " " << b << endl;  // 结果：1 0。因为 b 未赋值
```

8.5 逗号表达式

逗号表达式用于连接两个表达式，所有运算符中优先级最低。

用法示例：表达式 1，表达式 2，... 表达式 n

先求表达式 1，再求表达式 2……一直到表达式 n，整个表达式的值为表达式 n 的值。

```
x = (a = 3, 6 * 3);     // x 的值为 18
x = a, 6 * 3;  // x 的值为 3，赋值运算符优先级高于逗号运算符
```

if 语句比条件运算符可读性更高。

强制类型转换后，被转换的量的类型并没有发生改变。

8.6 位运算

运算符	含义	备注
&	按位与	双目
|	按位或	双目
^	按位异或	双目
~	取反	单目
<<	左移	单目
>>	右移	单目

```
a = ~a;          // 对 a 按位取反
a = a << 1;      // a 左移 1 位，相当于乘 2
a = a >> 1;      // a 右移 1 位，相当于除以 2
```

注意：

1. 右移时，无符号数左边高位移入 0；有符号数，原符号位为 0 时，左边移入 0，原符号位为 1 时，取决于编译器。
2. 不同长度的数进行位运算时，按右端对齐。如果短数是无符号整型，左侧补满 0；如果短数为有符号整型，则短数为正时左侧补满 0，为负时左侧补满 1。

用途：

1. 和 0 按位与，“清零”
2. 和特定数按位与，取指定位，如 `a & 377` 可以取出 a 的低 8 位
3. 按位与指定某些位为 1
4. 异或：与 1 异或，使特定位翻转；与 0 异或，使特定位保持不变

示例：a = 3，b = 4，不使用第三个变量，交换 a 和 b 的值。
解答：`a = a ^ b; b = b ^ a; a = a ^ b;`

9 C 语言的控制成分

任何具有单入口单出口的程序均可以用三种基本的结构表达：顺序结构、分支结构、循环结构。

`if` 语句的括号内可以是任意的数值类型。若括号内表达式的值为 0，按“假”处理；若非 0，按“真”处理。

`switch` 语句中，`default` 后也要跟 `break`。`default` 与 `case` 相比并没有那么特殊。
示例程序：

```
switch(3) {
    case 0:
        ...
    default:
        ...
    // default 后表面的 case 也会被执行
    case 1:
        ...
    case 2:
        ...
}
```

`for` 循环中嵌套 `switch` 语句，`switch` 中有 `break`，跳出哪一个？经过实验，`switch` 语句中的 `break` 仅仅跳出了 `switch`，没有跳出外部的循环。而如果把 `switch` 语句换成 `if` 语句，其中的 `break` 可以跳出外部循环。

在循环嵌套时，必须先跳出内层的循环，才能回到外层的循环。
示例程序：

```
int i = 0;
while (i < 2) {
    while (i < 5) {
        i++;
    }
}
cout << "i = " << i << endl;     // 输出结果为 i = 5
```

10 C 程序中的数组

10.1 数组的定义

定义数组：类型数组名 [常量表达式];
VC++ 不允许用变量定义数组的初始长度，但是 gcc/g++ 允许。
在 VC++ 中，如果想方便地修改数组的初始长度，有下面有种做法：

```
// 方法 1
const int i = 4;
int a[i] = {1, 2, 3, 4};
// 方法 2
#define N 4
int main()
{
    int a[N] = {1, 2, 3, 4};
    return 0;
}
```

10.2 数组的初始化

```
int a[] = {1, 2, 3, 4};          // 初始化后，数组 a 的长度为 4
int a[4] = {1, 2};               // 初始化后，a[0] 为 1，a[1] 为 2，其余为 0
int a[4] = {0};                  // 初始化后，a 的所有元素均为 0
int a[4] = {1, 2, 3, 4, 5, 6}; // 编译错误，数组下标越界
int a[][4] = {{1}, {0, 6}, {0, 0, 11}};     // 空缺部分为 0
int a[3][4] = {0};               // 指明数组维数时，将所有元素初始化为 0
```

10.3 数组的应用

打印数组：cout << setw(3);   // 输出控制符 setw(x)，设置输出占 x 个字宽
当有一些数据要存储时，用于存放一系列数据类型相同的数据。当处理的对象是连续的整数时，利用数据和下标间的对应关系，用于数字统计等应用问题。

```
double sqrt(double x);          // sqrt() 函数的参数和返回值都是 double 类型
```

11 字符数组和字符串

11.1 定义

```
char a[10] = {'a', 'b', 'c'};           // a[] 的后 7 位会被初始化为 '\0'
char a[] = {'C', 'h', 'i', 'n', 'a'};  // 初始化后，a[] 包含 5 个元素
char a[] = "China";     // 初始化后，a[] 包含 6 个元素，a[5] 为 '\0'
char a[5] = "China";    // 错误
```

内容相同的字符串会比字符数组多一个 '\0'。所以有 '\0' 结尾的字符数组都可以看作是字符串。

```
char a[6] = "China";    // 只可以在数组定义并初始化时可以用
```

不能用赋值语句将一个字符串常量或字符数组直接赋给另一个字符数组。

```
str1[] = "China";       // 赋值不合法
str1 = "China";         // 赋值不合法
str2 = "China";         // 赋值不合法
cout << "String 1: " << str1 << endl; // 打印字符数组
```

11.2 输入

键盘输入的内容在敲回车键后会先存到输入缓存区，然后再被程序读取。cin 会跳过空格和回车，把它们作为区分不同输入数据的标志。

```
float grade;
// 连续读入数据，如果不能读到 float 型数据，跳出循环。
while (cin >> grade) {
    ...
}
```

在用 cin 连续读取字符时，可以用 ^Z（Ctrl+Z）结束输入。

比较 cin.get()、cin.get(char) 和 getchar()

```
char c;
// cin.get() 会从输入缓存区读取一个字符，能读取空格字符和回车字符。也就是 cin.get() 说不会跳过空格
// 当作普通字符对待。EOF 是文件结束标志，表示输入结束。
while ((c = cin.get()) != EOF) {
    ...
}

// 下面的 cin.get(char) 与上面的 cin.get() 运行结果完全相同
while(cin.get(c)) {
    ...
}
```

```
// getchar() 不跳过任何字符
while (c = getchar()) {
    ...
}
```

使用 cout 打印字符数组时需要注意，字符数组要以 '\0' 结尾，否则会一直向后打印，直至遇到 '\0'。

```
char a[8] = {'H', 'e', 'l', 'l', 'o'};
cout << a;  // a[] 没有以'\0'结尾，会一直向后输出，直到遇到'\0'
```

输入字符串的三种方法：

1. cin
2. cin.get(ch[], length, end_char) （end_char 一般默认为'\n'）
3. cin.getline(ch[], length, end_char)

getline 和 get 的区别：getline 遇到终止标志字符时结束，缓存区指针移动到终止标志字符之后；get 遇到终止标志字符时停止读取，指针不移动。